Steven Lee, Allister Liu, Amy Leong

Operating Systems Problem Set 6

1) The following pseudo-code represents an attempt to enforce mutual exclusion in a critical region

```
void f()
{
/* DISABLE_INTERRUPTS is a generic name for an assembly-level
instruction that saves the current hardware interrupt mask
and disables all interrupts */
existing_mask=DISABLE_INTERRUPTS;
/* Perform critical region operation */
ENABLE_INTERRUPTS(existing_mask); /* Restore prev IRQ mask */
}
```

Discuss why is (or why is not) this approach valid for the following situations:

a) user-level code, single-threaded, single CPU

This approach is not valid. To enforce mutual exclusion in a critical region for this case, all signals should be blocked from entering the critical region, and then the previous signal mask should be restored upon exit. In the pseudo-code, all interrupts are blocked, not signals and the unblocked signals can still affect the critical region. (Interrupts cannot be disabled with user-level code but this is for the case that if it “can”).

b) user-level code, multi-threaded, single CPU

This approach is not valid. Interrupts are usually not sent all the way to user-level, therefore interrupts cannot be disabled. Even if the disable interrupt goes through or looking at it in the context of blocking all signals, due to multi-threaded CPU, other threads can still have access to the critical region.

c) user-level code, multi-threaded, multi-CPU

This approach is not valid, mostly the same reasoning as part (b), where the interrupts does not stop the other threads or in this case **multi-process** from accessing the critical region.

d) kernel-level code, single CPU

This approach is valid. Mutual exclusion is enforced for kernel-level code with a single CPU by masking all interrupts on entry and then restoring the mask on exit. The pseudo-code follows this by saving the current hardware interrupt mask, disabling all interrupts, and restoring the mask.

e) kernel-level code, multi-CPU

This approach is not valid because it is possible for a routine to run on another processor at the same time. Only the local processor is affected when interrupts are disabled.